Dashboard Design by AppWizard

Example Code Introduction for 64-bit NuMicro® Family

Document Information

Application	This example code demonstrates how to use the SEGGER emWin AppWizard tool to design a graphical user interface (GUI) for dashboard interface. It integrates the function of controlling LEDs and buttons on the MA35H0 series evaluation board.
BSP Version	emWin Software Package for MA35 Family
Hardware	NuMaker-HMI-MA35H04F70 V1.0

The information described in this document is the exclusive intellectual property of Nuvoton Technology Corporation and shall not be reproduced without permission from Nuvoton.

Nuvoton is providing this document only for reference purposes of NuMicro microcontroller and microprocessor based system design. Nuvoton assumes no responsibility for errors or omissions.

All data and specifications are subject to change without notice.

For additional information or questions, please contact: Nuvoton Technology Corporation.

www.nuvoton.com

1. Overview

This example will demonstrate how to use the SEGGER emWin AppWizard tool and the NuMaker-HMI-MA35H0-A1 evaluation board (EVB) to complete the graphical user interface (GUI) design. Taking the dashboard system interface as an example, the interface design and human-machine interaction functions. The AppWizard API functions are employed to link LED and button event processing, with real-time status updates displayed on the GUI. The AppWizard API functions help reduce development time.

1.1 UI Design

This example code uses the Nuvoton GUI development platform NuMaker-HMI-MA35H0-A1 evaluation board, which uses a 7" TFT-LCD resistive touch panel (with a resolution of 1024x600). In the project, the GUI design uses the SEGGER emWin AppWizard tool, (V1.40f_6.34g). The IDE development is divided into different parts: the Objects, Hierarchic Tree, Resource and Properties, as shown in Figure 1-1. Users can design the UI layout by dragging and dropping components from the Objects onto the Editor window.

Figure 1-1 SEGGER emWin AppWizard IDE

For setting up the Dashboard project, user can refer to its configuration settings as shown in Figure 1-2. With the NuMaker-HMI-MA35H0-A1 evaluation board, set the Display size to 1024 x 600 pixels. Set the Color format to “32bit, GUICC_M8888I” and enable “Enable Multibuffering”.

The architecture diagram is shown in Figure 1-3. The GUI design is divided into three pages: Initialization Screen, Login Screen and Dashboard Screen.

Initialization Screen	Display the start page and set the timer to jump to the login screen after 3 seconds.
Login Screen	Display the user login screen, enter the password, and if it is correct, you will be logged in and jump to the Dashboard Screen. Default Password: 1234
Dashboard Screen	The function demonstration page showcases the current status of the EVB through the interactive control of the onboard LEDs and buttons, with real-time updates displayed on the screen.

Table 1-1 GUI Architecture Table

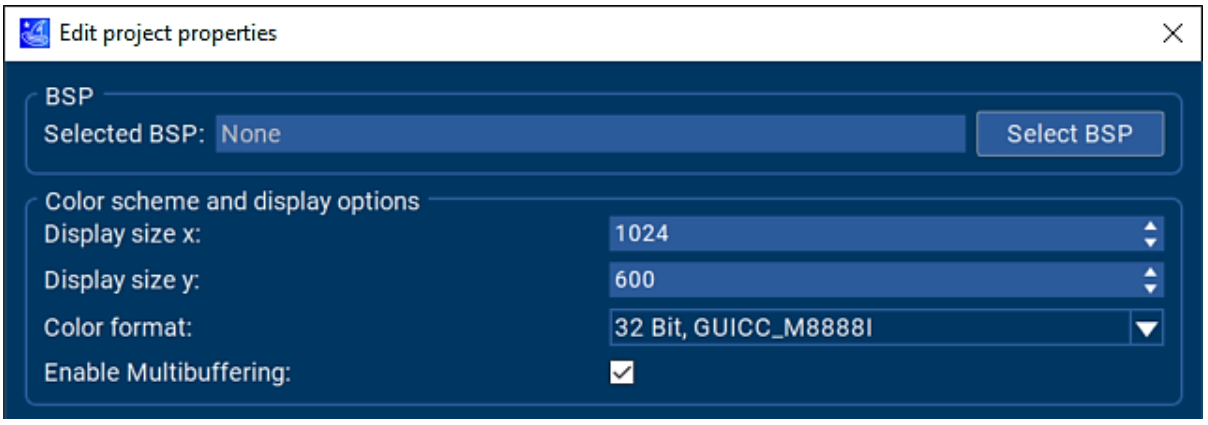

Figure 1-2 The Dashboard Project's Properties

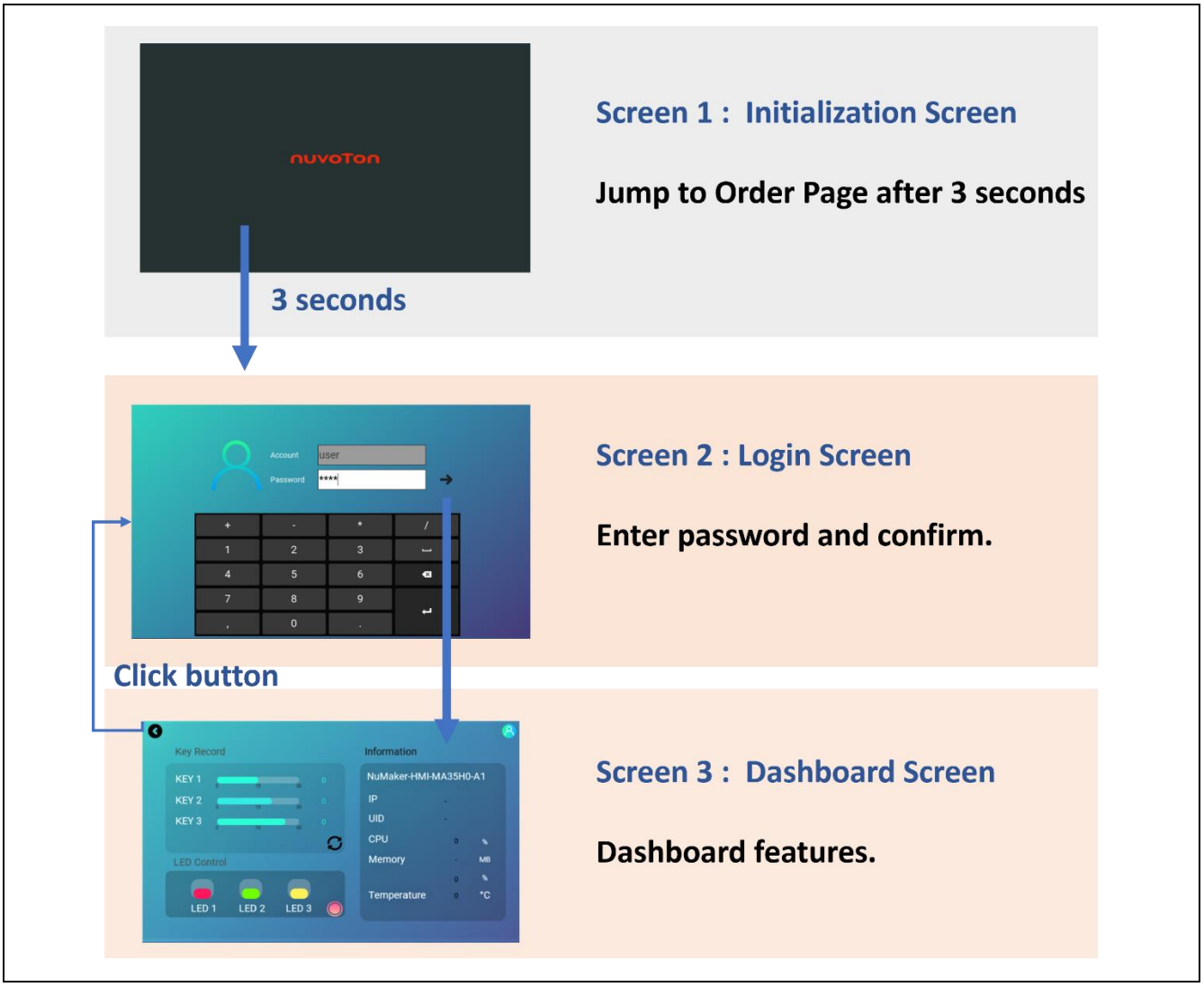

Figure 1-3 Flowchart of the Dashboard Project

In the design of the Initialization Screen, the layout is configured as shown in Figure 1-4. The objects used include Screen, Box, and Image. The Screen size is 1024 x 600 pixels, and the background color is set using the Box object with the color values Red = 38, Green = 50, Blue = 50, and Alpha = 255.

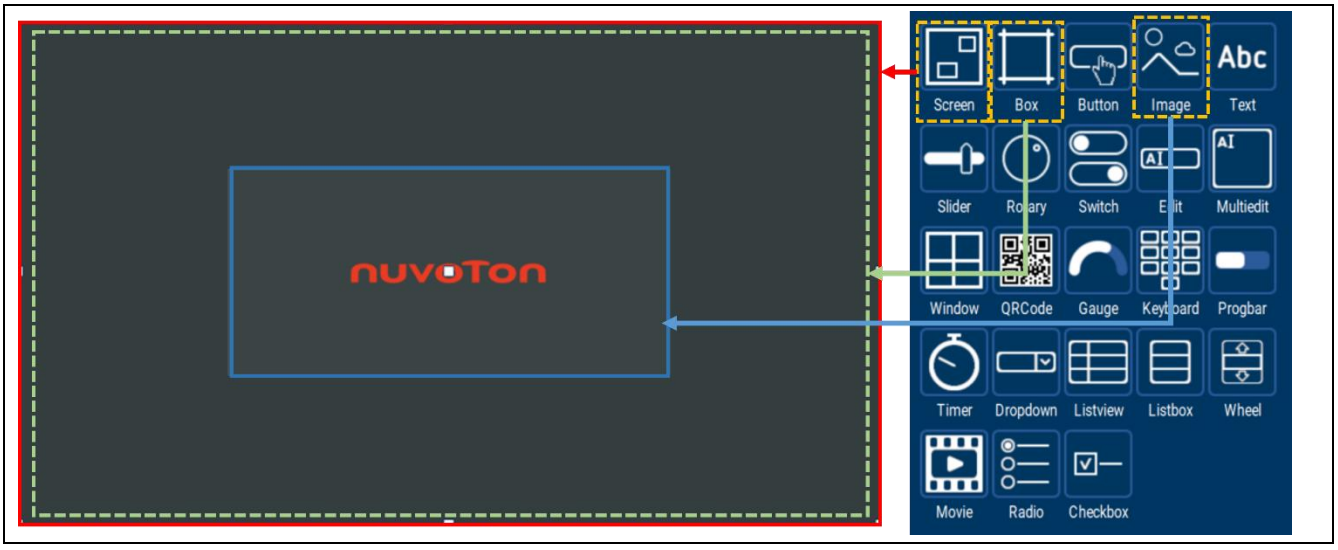

Figure 1-4 Initialization Screen Design

The Login Screen is designed as a user login interface and the layout is configured as shown in Figure 1-5. The account and password use the Edit object with the maximum data length set to 10. The keyboard uses the keyboard object and is set to SKEYBOARD_NUMPAD, displaying only the numeric keypad. An Image object is used as the background, and the Image size is set to 1024 x 600 pixels.

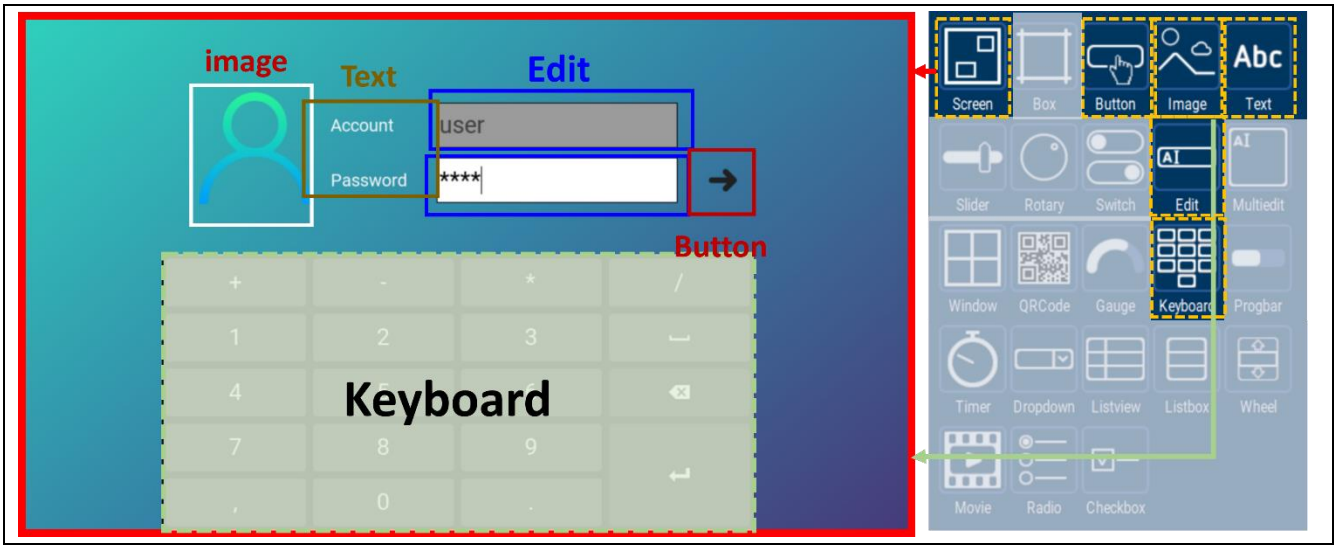

Figure 1-5 Login Screen Design

The Dashboard Screen consists of buttons and a program bar, with information fields arranged sequentially using text objects. An image is used as the background, set to 1024 x 600 pixels. The layout of the GUI objects is configured as shown in Figure 1-6.

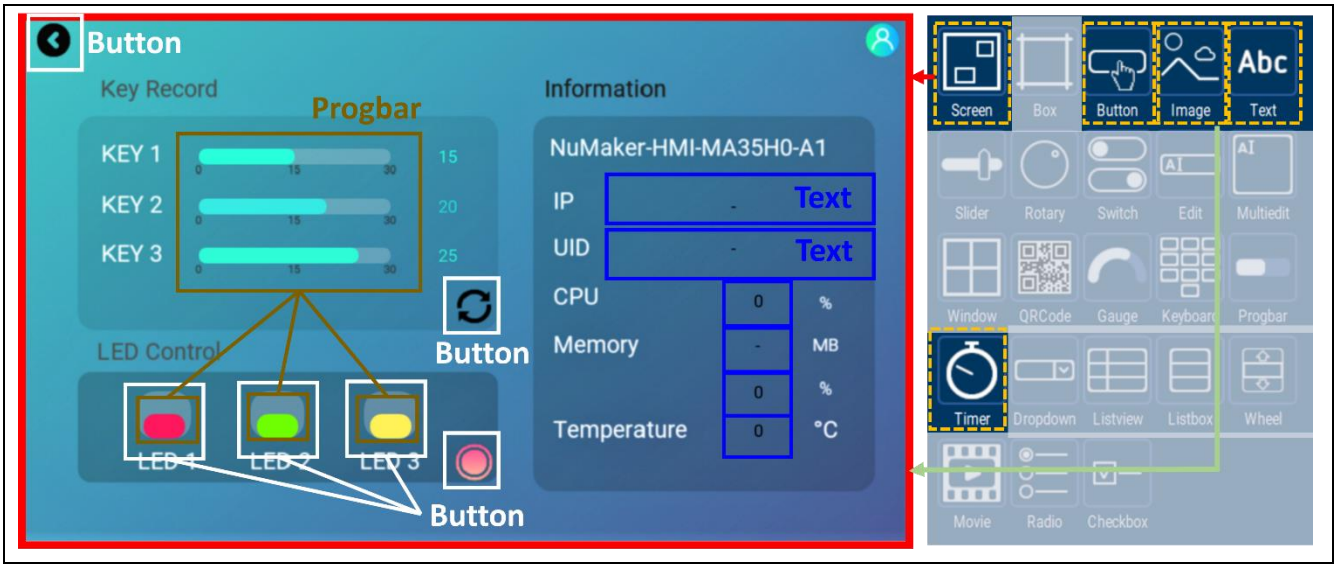

Figure 1-6 Dashboard Screen Design

1.2 UI Feature Design

The emWin AppWizard provides the "**Interactions**" function. Interactions can define user behavior operations and establish links between objects, such as automatic animation design or sliding transitions to switch between screens. The design interaction of this project is as follows.

The Initialization Screen is set with a timer of 3000 milliseconds, after which it will transition to the Login Screen. The Interactions are configured as shown in Figure 1-7. Once the Screen initialization is complete, the timer will start counting down. After the three-second countdown finishes, it will trigger the SHOWSCREEN Job to transition to the next screen.

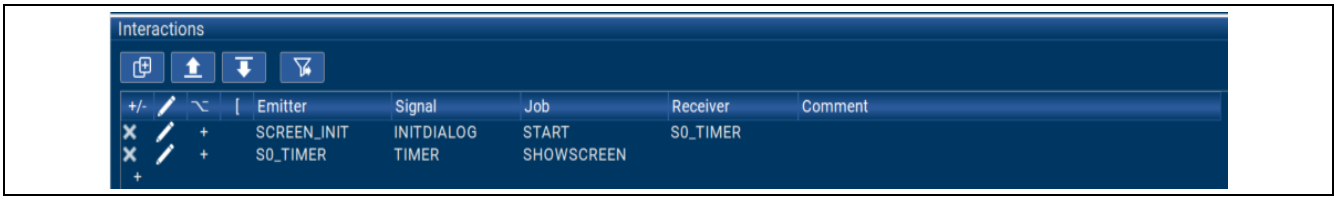

Figure 1-7 Interactions Settings of Initialization Screen

When the Screen is created, the default values for the account and password are automatically filled in. The account is set to 'user' with the enable state set to 'off', making it unchangeable by the user. The password is set to the default value '1234'. After pressing the S1_BUTTON_LOGIN button, a function is called that checks the correctness of the password. If the password is correct, the screen transitions to the next page; if incorrect, the S1_EDIT_PASSWORD is cleared.

Figure 1-8 Interactions Settings of Login Screen

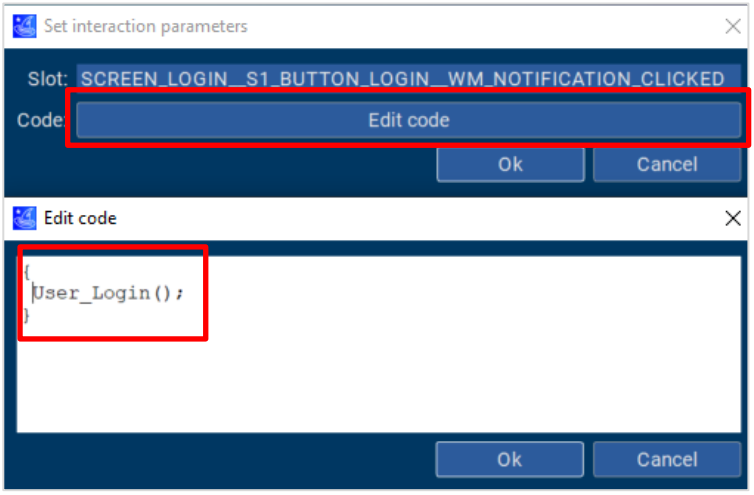

Figure 1-9 Setting Call Function Settings for Key Event

When the Dashboard Screen is created, it reads the evaluation board's system information to update the IP and UID. The Interactions for Key Control and LED Control are configured as shown in Figure 1-10. The buttons on the GUI interface are configured to turn on/off the LEDs on the evaluation board, and vice versa, the keys on the evaluation board trigger the counting function on the GUI interface. The Dashboard Screen is set with a timer of 2000 milliseconds, which updates the system values every 2 seconds, including CPU loading, Memory, and Temperature values.

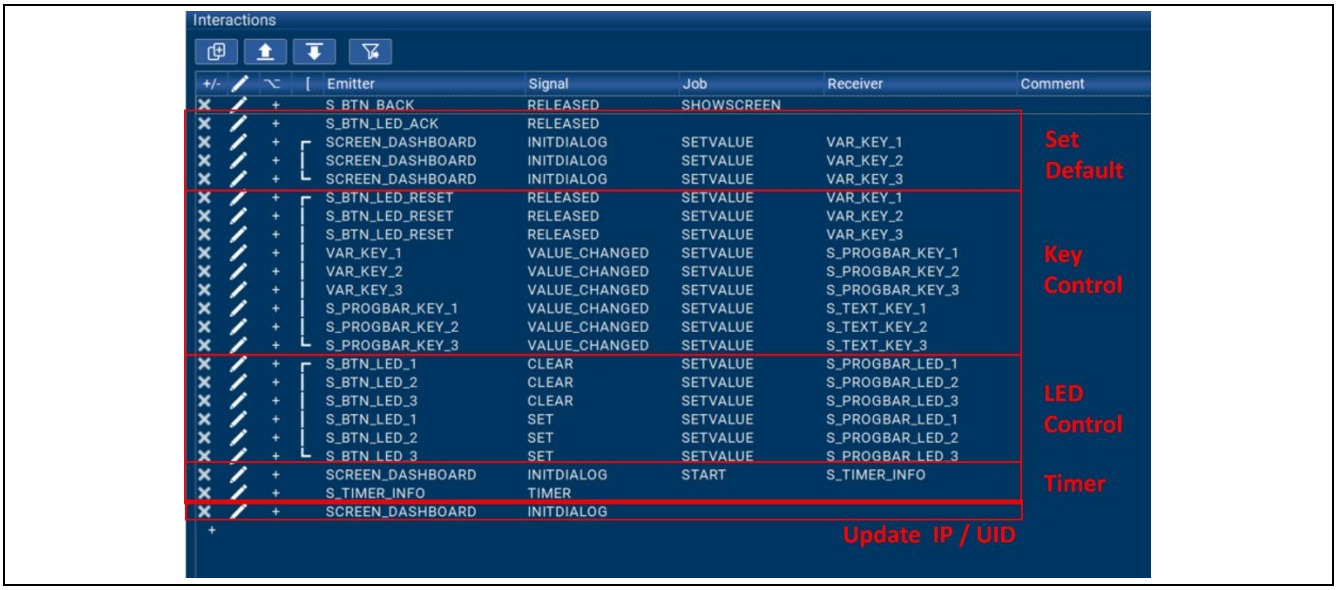

Figure 1-10 Interactions Settings of Dashboard Screen

Once the GUI design is complete, the AppWizard tool will automatically convert the GUI design into code. The user should click **File → Export & Save** to convert it into the corresponding code, as shown in Figure 1-11. The converted code will be placed in the Source and Resource folders.

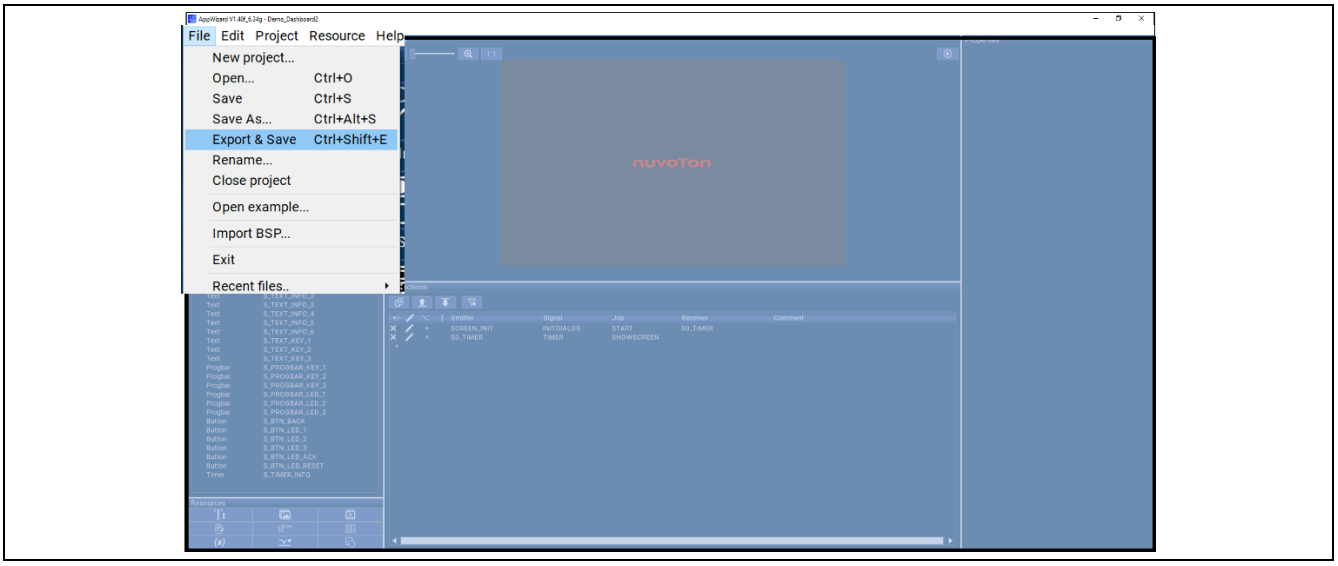

Figure 1-11 Export & Save of AppWizard Tool

1.3 Program Compilation

After the UI design is completed, the code needs to be compiled. Once compilation is complete, an executable file named *SimpleDemoAppWizard* will be generated, which can run on the NuMaker-HMI-MA35H0-A1 development platform. Using the NuMaker-HMI-MA35H0-A1 as an example, the following explains the compilation process in a Linux environment:

1. Move the MA35_Family_Linux_emWin_Package folder into the MA35H0 Linux environment.
2. Overwrite the Source and Resource folders exported from the AppWizard project into the MA35_Family_Linux_emWin_Package > Sample > Dashboard > Application folder. During the cross-compilation process, the GUI code will be compiled together with the project into an executable file.
3. Then running cross-compilation, users need to pay attention to the toolchain settings. In the build configuration, manually modify the CC and STRIP fields in the Makefile to set them to the MA35H0 toolchain:

```
CC=~/MA35D1_Buildroot/output/host/bin/aarch64-nuvoton-linux-gnu-gcc
STRIP=~/MA35D1_Buildroot/output/host/bin/aarch64-nuvoton-linux-gnu-strip
```

If the toolchain files are missing, execute the MA35H0 image compilation and refer to the MA35H0 Buildroot Project User Manual.

4. Compile using the make command:

```
~/MA35_Family_Linux_emWin_64bit_Package/Sample/Dashboard$ make
```

To clear the compiled files or recompile, you need to execute the following commands:

```
~/MA35_Family_Linux_emWin_64bit_Package/Sample/Dashboard$ make clean
```

For information on Linux compilation, please refer to the MA35H0_Linux User Manual.

5. Once the compilation is complete, an executable file named *SimpleDemoAppWizard* will be generated, which can run on the NuMaker-HMI-MA35H0-A1 evaluation board.
6. Users can use a USB drive to transfer the files. Place the executable file (*SimpleDemoAppWizard*) and the scripts (getDeviceInfo.sh and getSysMonitor.sh) onto the USB drive. After the NuMaker-HMI-MA35H0-A1 boots up, run the mount command:

```
$ mount /dev/sda1 /mnt
```

Move to the /mnt folder:

```
$ cd /mnt
```

Run executable file:

```
$ ./SimpleDemoAppWizard
```

If running the NuMaker-HMI-MA35H0-A1 for the first time, screen calibration points need to be set. Copy ts.conf to the development platform using a USB drive:

```
$ cp /mnt/ts.conf /etc/
```

Run the ts_calibrate command to perform a five-point screen calibration:

```
$ /usr/bin/ts_calibrate
```

2. Code Description

After converting the GUI design into the corresponding code, users can add custom function features in *~/Application/Source/CustomCode/Application.c*. At the beginning of the program, users need to include the pages that will be used. This code uses the AppWizard API. For reference, please consult the AppWizard User Guide & Reference Manual. The table below lists the AppWizard API functions that will be used in this example code.

Routine	Description
APPW_GetText()	This function stores the text of an object in the given buffer.
APPW_SetText()	This function sets a text to an object.
APPW_GetValue()	This function sets the value of an object.
APPW_SetValue()	This function returns the value of an object.

Table 2-1 AppWizard API Table

```
// Main application header file
#include "Application.h"

// Header for dashboard screen generation
#include "Application/Source/Generated/SCREEN_DASHBOARD.h"
#include "Application/Source/Generated/SCREEN_LOGIN.h"
#include "Application/Source/Generated/Resource.h

// Path to the input device (e.g., keyboard or mouse)
#define DEVICE "/dev/input/event1"

// Pointer to an integer that stores an error code
int *error;
```

The LED control code is as follows. It determines the LED's on/off behavior based on the current GUI LED state. If the current LED state is on, pressing the button will turn off the LED; conversely, if the LED is off, pressing the button will turn it on:

```
/*=============================================================================================*
** Function : Control LED on and off
** Output   : None
** use      : LED_Toggle(1);
**=============================================================================================*/
void LED_Toggle(char led) {

    int status;
    char command[256];
    char meg[256];
```

```
    // Check the status of the specified LED
    if (led == 0) {
        status = APPW_GetValue(SCREEN_DASHBOARD, S_PROGBAR_LED_1, error);
    } else if (led == 1) {
        status = APPW_GetValue(SCREEN_DASHBOARD, S_PROGBAR_LED_2, error);
    } else if (led == 2) {
        status = APPW_GetValue(SCREEN_DASHBOARD, S_PROGBAR_LED_3, error);
    } else {
        // Invalid LED number, log an error message and return
        printf("[E] Invalid LED number %d \n", led + 1);
        return;
    }

    // Toggle the LED state based on its current status
    if (status == 100) {
        // If LED is on (status 100), turn it off
        snprintf(command, sizeof(command), "echo 0 > /sys/class/leds/LED%d/brightness",
led);
        // Execute the command to toggle the LED
        system(command);
        printf("[L] Control LED%d OFF ! \n", led + 1);

    } else {
        // If LED is off, turn it on
        snprintf(command, sizeof(command), "echo 1 > /sys/class/leds/LED%d/brightness",
led);
        // Execute the command to toggle the LED
        system(command);
        printf("[L] Control LED%d ON ! \n", led + 1);
    }
}
```

The button reading program on the evaluation board is as follows. It starts a thread that continuously monitors whether the button on the evaluation board is pressed. When the button is pressed, it updates the count displayed on the GUI.

```
/**
 * Thread function to monitor button events.
 * This function continuously reads input events from a device and updates
 * the application variables based on specific key events.
 */

void* button_event(void* arg) {
    int fd, count = 0;
    struct input_event ev;

    // Open the input device for reading
    fd = open(DEVICE, O_RDONLY);
    if (fd == -1) {
        printf("[E] Failed to open device \n");
        pthread_exit(NULL);  // Use pthread_exit instead of return
    }

    printf("[!] Monitoring key events...\n");
    while (1) {
        // Read input event from the device
```

```
        ssize_t bytesRead = read(fd, &ev, sizeof(struct input_event));

        if (bytesRead < sizeof(struct input_event)) {
            printf("[E] Failed to read input event \n");
            close(fd);
            pthread_exit(NULL);
        }

        // Process the key event
        if (ev.type == EV_KEY) {
            // Update the application variable based on the key code
            if (ev.code == 105 && ev.value == 1) {
                count = APPW_GetVarData(VAR_KEY_1, error) + 1;
                APPW_SetVarData(VAR_KEY_1, count);
            }

            … …
        }
    }

    close(fd);
    pthread_exit(NULL);  // End thread
}
```

When the program starts running, it obtains the system information from the evaluation board by reading a script. The code is as follows:

```
/*=============================================================================================*
** Function : Device information update
** Output   : 0 | 1: error
** use      : UpdateDeviceInfo();
**=============================================================================================*/

int UpdateDeviceInfo(void) {
    FILE *fp;
    char output[100];
    char IP[256], UUID[256];
    char *delimiter;
    char meg[256];

    // Run the shell script to get device information
    fp = popen("./getDeciceInfo.sh", "r");
    if (fp == NULL)  return 1;

    … …

    pclose(fp);

    // Remove the newline character from the output
    output[strcspn(output, "\n")] = '\0';

    // Split the output string using ';' as a delimiter
    delimiter = strchr(output, ';');
    if (delimiter != NULL) {
        *delimiter = '\0';
```

```
        // Get UUID text
        strncpy(UUID, delimiter + 1, sizeof(UUID) - 1);
        // Get IP text
        strncpy(IP, output, sizeof(IP) - 1);
        // Clean up the strings
        IP[sizeof(IP) - 1] = '\0';
        UUID[sizeof(UUID) - 1] = '\0';

        // Update the screen with the new IP and UUID
        APPW_SetText(SCREEN_DASHBOARD, S_TEXT_INFO_1, IP);
        APPW_SetText(SCREEN_DASHBOARD, S_TEXT_INFO_2, UUID);

        … …
    }
    … …

    return 0;
}
```

This example code obtains the IP and UID information from the evaluation board via the command line. The getDeviceInfo.sh script is as follows:

```
#!/bin/sh

# Get the IP address for eth0
eth0_IP="$(ifconfig | grep -Eo 'inet (addr:)?([0-9]*\.){3}[0-9]*' | grep -Eo '([0-
9]*\.){3}[0-9]*' | grep -v '127.0.0.1')"

# Read memory addresses and concatenate them to form UID
uid_part0="$(devmem 0x40460180)"
uid_part1="$(devmem 0x40460184)"
uid_part2="$(devmem 0x40460188)"
UID="${uid_part0#0x}${uid_part1#0x}${uid_part2#0x}"

# Print the eth0 IP and the UID variable
echo "${eth0_IP};${UID}"
```

This example code obtains CPU loading, memory usage, and temperature information from the evaluation board via the command line. The getSysMonitor.sh script is as follows:

```
#!/bin/sh

# Define a function to read the system status
sys_status() {
  # Output the CPU usage
  echo -n "$(cpu_usage);"

  # Output the RAM usage
  echo -n "$(ram_usage);"

}

# Retrieve CPU usage
cpu_usage() {
  # Get the CPU idle value
  cpu=$(top -bn1 | grep "CPU:" | grep -v "grep" | awk '{ print $2}' | sed 's/%//')
```

```
  # Get the user value
  usr=$(top -bn1 | grep "CPU:" | grep -v "grep" | awk '{ print $4}' | sed 's/%//')

  # Output the sum of CPU and user values
  echo $((cpu + usr))
}

# Retrieve RAM usage
ram_usage() {
  free -m | awk '/Mem:/ { printf("%d/%d MB (%.2f%%)", $3, $2, $3/$2 * 100) }'
}


# Output information
# Decide which function to execute based on the argument
case "$1" in
  "Sys Status")
    sys_status
    ;;
  *)
    sys_status
    ;;
esac
```

For temperature information, please refer to MA35H0_Linux User Manual. When setting up the MA35H0 development platform, users need to enable the temperature sensor functionality:

```
Device Drivers --->
  [*] Thermal drivers --->
    [*] Expose thermal sensors as hwmon device
    [*] APIs to parse thermal data out of device tree
    [*] Enable writable trip points
    [*] User_space thermal governor
    [*] Generic device cooling support
          Nuvoton thermal drivers --->
              <*> MA35H0 Temperature Sensor
```

After enabling it, you can obtain the evaluation board's temperature information via the command line:

```
$ cat /sys/class/thermal/thermal_zone0/temp
58
```

3. Software and Hardware Requirements

3.1 Software Requirements

- Version of emWin AppWizard IDE
  - AppWizard_V1.40f_6.34g

3.2 Hardware Requirements

- Circuit components
  - NuMaker-HMI-MA35H0-A1 V1.0

4. Directory Information

The directory structure is shown below.

EC_MA35_AppWizard_Dashboard_V1.00	
AppWizard_Project	
Firmware	Program scripts and executable file
Resource	Image files used for this example
Source	Source file of example code
Simulation	AppWizard simulation file
MA35_Family_Linux_emWin_Package	
Config	LCD configuration file
Include	GUI related function library
Lib	AppWizard library file
Sample	
Dashboard	
Application	
Resource	Image files used for this example
Source	Source file of example code

Figure 4-1 Directory Structure

5. Example Code Execution

1. Copy the executable and scripts from the AppWizard_Project → Firmware folder to the NuMaker-HMI-MA35H0-A1 evaluation board.
   - Executable: EC_MA35_Dashboard
   - Scripts*: getDeviceInfo.sh, getSysMonitor.sh*
   - For instructions on how to move the executable files, refer to Section 1.3, item 6.
2. Run the executable EC_MA35_Dashboard.

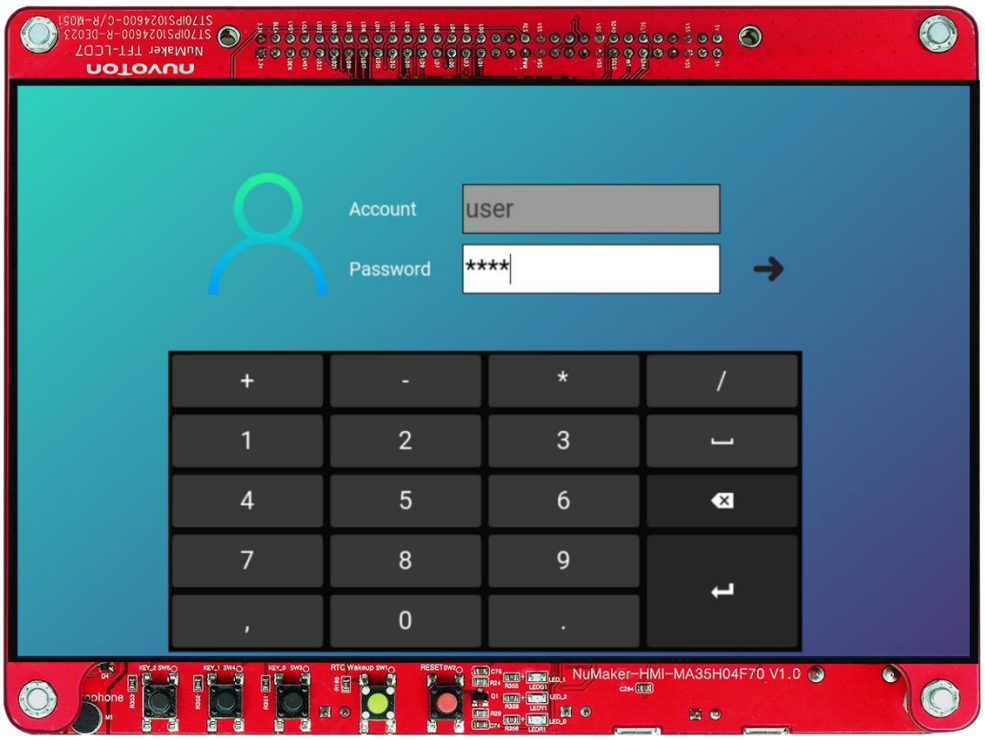

Figure 5-1 Execute EC_MA35_ Dashboard Result

6. Revision History

Date	Revision	Description
2024.09.10	1.00	Initial version.

Important Notice

Nuvoton Products are neither intended nor warranted for usage in systems or equipment, any malfunction or failure of which may cause loss of human life, bodily injury or severe property damage. Such applications are deemed, "Insecure Usage".

Insecure usage includes, but is not limited to: equipment for surgical implementation, atomic energy control instruments, airplane or spaceship instruments, the control or operation of dynamic, brake or safety systems designed for vehicular use, traffic signal instruments, all types of safety devices, and other applications intended to support or sustain life.

All Insecure Usage shall be made at customer's risk, and in the event that third parties lay claims to Nuvoton as a result of customer's Insecure Usage, customer shall indemnify the damages and liabilities thus incurred by Nuvoton.

Please note that all data and specifications are subject to change without notice.
All the trademarks of products and companies mentioned in this datasheet belong to their respective owners.